The main directives

The FastCGI, uWSGI, and SCGI modules are included in the default Nginx build. You do not need to enable them manually at compile time. The directives listed in the following table allow you to configure the way Nginx *passes* requests to the FastCGI/uWSGI/SCGI application. Note that you will find the `fastcgi_params`, `uwsgi_params`, and `scgi_params` files, which define the directive values that are valid for most situations, in the Nginx configuration folder:

Directive	Description

`fastcgi_pass`
Context: `location`, `if`
	This directive specifies that the request should be passed to the FastCGI server by indicating its location:
	• For TCP sockets, the syntax is: `fastcgi_pass hostname:port;`
	• For Unix domain sockets, the syntax is: `fastcgi_pass unix:/path/to/fastcgi.socket;`
	• You may also refer to upstream blocks (read the following sections for more information): `fastcgi_pass myblock;`
	Examples:
	`fastcgi_pass localhost:9000;`
	`fastcgi_pass 127.0.0.1:9000;`
	`fastcgi_pass unix:/tmp/fastcgi.socket;`
	`# Using an upstream block`
	`upstream fastcgi {`
	`  server 127.0.0.1:9000;`
	`  server 127.0.0.1:9001;`
	`}`
	`location ~* \.php$ {`
	`  fastcgi_pass fastcgi;`
	`}`

Directive	Description

`fastcgi_param`
Context: `http`, `server`, `location`
	This directive allows you to configure the request passed to FastCGI. Two parameters are strictly required for all FastCGI requests: `SCRIPT_FILENAME` and `QUERY_STRING`.
	Example:
	`fastcgi_param SCRIPT_FILENAME /home/website.com/www$fastcgi_script_name;`
	`fastcgi_param QUERY_STRING $query_string;`
	As for POST requests, additional parameters are required: `REQUEST_METHOD`, `CONTENT_TYPE`, and `CONTENT_LENGTH`:
	`fastcgi_param REQUEST_METHOD $request_method;`
	`fastcgi_param CONTENT_TYPE $content_type;`
	`fastcgi_param CONTENT_LENGTH $content_length;`
	The `fastcgi_params` file that you will find in the Nginx configuration folder already includes all of the necessary parameter definitions (except for the `SCRIPT_FILENAME`), which you need to specify for each of your FastCGI configurations.
	If the parameter name begins with `HTTP_`, it will override the potentially existing HTTP headers of the client request.
	You may optionally specify the `if_not_empty` keyword, forcing Nginx to transmit the parameter only if the specified value is not empty.
	Syntax: `fastcgi_param PARAM value [if_not_empty];`

`fastcgi_bind` Context: `http`, `server`, `location`	This directive binds the socket to a local IP address, allowing you to specify the network interface you want to use for FastCGI communications. Syntax: `fastcgi_bind IP_address;`
`fastcgi_pass_header` Context: `http`, `server`, `location`	This directive specifies the additional headers that should be passed to the FastCGI server. Syntax: `fastcgi_pass_header headername;` Example: `fastcgi_pass_header Authorization;`

Directive	Description
`fastcgi_hide_header` Context: `http`, `server`, `location`	This directive specifies the headers that should be hidden from the FastCGI server (headers that Nginx does not forward). Syntax: `fastcgi_hide_header headername;` Example: `fastcgi_hide_header X-Forwarded-For;`
`fastcgi_index` Context: `http`, `server`, `location`	The FastCGI server does not support automatic directory indexes. If the requested URI ends with a `/`, Nginx appends the value `fastcgi_index`. Syntax: `fastcgi_index filename;` Example: `fastcgi_index index.php;`
`fastcgi_ignore_client_abort` Context: `http`, `server`, `location`	This directive lets you define what happens if the client aborts their request to the web server. If the directive is turned on, Nginx ignores the abort request and finishes processing the request. If it's turned off, Nginx does not ignore the abort request. It interrupts the request treatment and aborts all related communication with the FastCGI server. Syntax: `on` or `off` Default: `off`
`fastcgi_intercept_errors` Context: `http`, `server`, `location`	This directive defines whether or not Nginx should process the errors returned by the gateway or directly return error pages to the client. (Note: Error processing is done via the `error_page` directive of Nginx.) Syntax: `on` or `off` Default: `off`
`fastcgi_read_timeout` Context: `http`, `server`, `location`	This directive defines the timeout for the response from the FastCGI application. If Nginx does not receive the response within this period, the `504 Gateway Timeout` HTTP error is returned. Syntax: Numerical value (in seconds) Default: 60 seconds

Directive	Description
`fastcgi_connect_timeout` Context: `http`, `server`, `location`	This directive defines the backend server connection timeout. This is different from the read/send timeout. If Nginx is already connected to the backend server, the `fastcgi_connect_timeout` is not applicable. Syntax: Time value (in seconds) Default: 60 seconds
`fastcgi_send_timeout` Context: `http`, `server`, `location`	This is the timeout for sending data to the backend server. The timeout isn't applied to the entire response delay but between two write operations. Syntax: Time value (in seconds) Default value: `60` seconds

`fastcgi_split_path_info`
Context: `location`
	A directive particularly useful for URLs of the following form: `http://website.com/page.php/param1/param2/`.
	The directive splits the path information according to the specified regular expression:
	`fastcgi_split_path_info ^(.+\.php)(.*)$;`
	This affects two variables:
	• `$fastcgi_script_name`: The filename of the actual script to be executed (in the example: `page.php`)
	• `$fastcgi_path_info`: The part of the URL that is after the script name (in the example: `/param1/param2/`)
	These can be employed in further parameter definitions:
	`fastcgi_param SCRIPT_FILENAME /home/website.com/www$fastcgi_script_name;`
	`fastcgi_param PATH_INFO $fastcgi_path_info;`
	Syntax: A regular expression

`fastcgi_store` Context: `http`, `server`, `location`	This directive enables a simple *cache store* where responses from the FastCGI application are stored as files on the storage device. When the same URI is requested again, the document is directly served from the cache store instead of forwarding the request to the FastCGI application. This directive enables or disables the cache store. Syntax: `on` or `off`

Directive	Description
`fastcgi_store_access` Context: `http`, `server`, `location`	This directive defines the access permissions applied to the files created in the context of the cache store. Syntax: `fastcgi_store_access [user:r|w|rw] [group:r|w|rw] [all:r|w|rw];` Default: `fastcgi_store_access user:rw;`
`fastcgi_temp_path` Context: `http`, `server`, `location`	This directive sets the path of temporary and cache store files. Syntax: The file path Example: `fastcgi_temp_path /tmp/nginx_fastcgi;`
`fastcgi_max_temp_file_size` Context: `http`, `server`, `location`	Set this directive to 0 to disable the use of temporary files for FastCGI requests or to specify a maximum file size. Default value: 1 GB Syntax: Size value Example: `fastcgi_max_temp_file_size 5m;`
`fastcgi_temp_file_write_size` Context: `http`, `server`, `location`	This directive sets the write buffer size when saving temporary files to the storage device. Syntax: Size value Default value: 8k or 16k
`fastcgi_send_lowat` Context: `http`, `server`, `location`	This option allows you to make use of the `SO_SNDLOWAT` flag for TCP sockets under FreeBSD only. This value defines the minimum number of bytes in the buffer for output operations. Syntax: Numerical value (size) Default value: `0`

`fastcgi_pass_request_body`
`fastcgi_pass_request_headers`
Context: `http`, `server`, `location`
	This directive defines whether or not the request body and extra request headers should be passed on to the backend server.
	Syntax: `on` or `off`;
	Default: `on`

Directive	Description
`fastcgi_ignore_headers` Context: `http`, `server`, `location`	This directive prevents Nginx from processing one or more of the following headers from the backend server response: • `X-Accel-Redirect` • `X-Accel-Expires` • `Expires` • `Cache-Control` • `X-Accel-Limit-Rate` • `X-Accel-Buffering` • `X-Accel-Charset` Syntax: `fastcgi_ignore_headers header1 [header2...];`

`fastcgi_next_upstream`
Context: `http`, `server`, `location`
	When `fastcgi_pass` is connected to an upstream block, this directive defines the cases where requests should be abandoned and re-sent to the next upstream server of the block. The directive accepts a combination of values among the following:
	• `error`: An error occurs while communicating or attempting to communicate with the server
	• `timeout`: A timeout occurs during transfers or connection attempts
	• `invalid_header`: The backend server returns an empty or invalid response
	• `http_500`, `http_502`, `http_503`, `http_504`, `http_404`: In case such HTTP errors occur, Nginx switches to the next upstream
	• `off`: Forbids the use of the next upstream server
	Examples:
	`fastcgi_next_upstream error timeout http_504;`
	`fastcgi_next_upstream timeout invalid_header;`

`fastcgi_next_upstream_timeout` Context: `http`, `server`, `location`	Defines the timeout to be used in conjunction with `fastcgi_next_upstream`. Setting this directive to `0` disables it. Syntax: Time value (in seconds)
`fastcgi_next_upstream_tries` Context: `http`, `server`, `location`	Defines the maximum number of upstream servers to be tried before returning an error message. This is to be used in conjunction with `fastcgi_next_upstream`. Syntax: Numerical value (default: 0)

Directive	Description
`fastcgi_catch_stderr` Context: `http`, `server`, `location`	This directive allows you to intercept some of the error messages sent to `stderr` (the Standard Error stream) and store them in the Nginx error log. Syntax: `fastcgi_catch_stderr filter;` Example: `fastcgi_catch_stderr "PHP Fatal error:";`
`fastcgi_keep_conn` Context: `http`, `server`, `location`	When set to `on`, Nginx will conserve the connection to the FastCGI server, thus reducing the overhead. Syntax: `on` or `off` (default: `off`). Note that there is no equivalent directive in the uWSGI and SCGI modules.
`fastcgi_force_ranges` Context: `http`, `server`, `location`	When set to `on`, Nginx will enable byte-range support on responses from the FastCGI backend. Syntax: `on` or `off` (default: `off`).
`fastcgi_limit_rate` Context: `http`, `server`, `location`	Allows you to limit the rate at which Nginx downloads the response from the FastCGI backend. Syntax: Numerical value (bytes per second)

FastCGI caching and buffering

Once you have correctly configured Nginx to work with your FastCGI application, you may optionally make use of the following directives, which will help you improve the overall server performance by setting up a cache system. Additionally, FastCGI buffering allows you to buffer the responses from the FastCGI backend instead of synchronously forwarding them to the client.

Directive	Description
`fastcgi_cache` Context: `http`, `server`, `location`	This directive defines a cache zone. The identifier given to the zone is to be reused in further directives. Syntax: `fastcgi_cache zonename;` Example: `fastcgi_cache cache1;`

Directive	Description
`fastcgi_cache_key` Context: `http`, `server`, `location`	This directive defines the cache key. In other words, it is what differentiates one cache entry from another. If the cache key is set to `$uri`, as a result, all requests with a similar `$uri` will correspond to the same cache entry. It's not enough for most dynamic websites; you also need to include the query string arguments in the cache key so that `/index.php` and `/index.php?page=contact` do not point to the same cache entry. Syntax: `fastcgi_cache_key key;` Example: `fastcgi_cache "$scheme$host$request_uri $cookie_user";`
`fastcgi_cache_methods` Context: `http`, `server`, `location`	This directive defines the HTTP methods eligible for caching. `GET` and `HEAD` are included by default and cannot be disabled. You may, for example, enable the caching of `POST` requests. Syntax: `fastcgi_cache_methods METHOD;` Example: `fastcgi_cache_methods POST;`
`fastcgi_cache_min_uses` Context: `http`, `server`, `location`	This directive defines the minimum number of hits before a request becomes eligible for caching. By default, the response of a request is cached after one hit (further requests with the same cache key will receive the cached response). Syntax: Numerical value Example: `fastcgi_cache_min_uses 1;`

Directive	Description
`fastcgi_cache_path` Context: `http`, `server`, `location`	This directive indicates the directory for storing the cached files as well as other parameters. Syntax: `fastcgi_cache_path path [levels=numbers] keys_zone=name:size [inactive=time] [max_size=size] [loader_files=number] [loader_sleep=time] [loader_threshold=time];` The additional parameters are: • `levels`: Indicates the depth of subdirectories (1:2 indicates that subfolders will be created down to two levels) • `keys_zone`: Selects the zone that you previously declared with the `fastcgi_cache` directive, and indicates the size to be occupied in memory • `inactive`: If a cached response is not used within the specified time frame, it's removed from the cache (default: 10 minutes) • `max_size`: Defines the maximum size of the entire cache • `loader_files, loaded_sleep`, and `loader_threshold`: Configures the cache loader: the amount of files it processes in one read cycle (`loader_files`, default: 100 files), the pause time between read cycles (`loader_sleep`, default: 50 ms), and the maximum duration of a read cycle (`loader_threshold`, default: 200 ms) Example: `fastcgi_cache_path /tmp/nginx_cache levels=1:2 zone=zone1:10m inactive=10m max_size=200M;`
`fastcgi_cache_use_stale` Context: `http`, `server`, `location`	This directive defines whether or not Nginx should serve stale cached data in certain circumstances (with regard to the gateway). If you use `fastcgi_cache_use_stale timeout`, and if the gateway times out, then Nginx will serve cached data. Syntax: `fastcgi_cache_use_stale [updating] [error] [timeout] [invalid_header] [http_500];` Example: `fastcgi_cache_use_stale error timeout;`

Directive	Description

`fastcgi_cache_valid`
Context: `http`, `server`, `location`
	This directive allows you to customize the caching time for different kinds of response code. You may cache responses associated to the `404` error codes for `1` minute, and in the opposite cache, `200 OK` responses for `10` minutes or more. This directive can be inserted more than once, demonstrated as follows:
	`fastcgi_cache_valid 404 1m;`
	`fastcgi_cache_valid 500 502 504 5m;`
	`fastcgi_cache_valid 200 10;`
	Syntax: `fastcgi_cache_valid code1 [code2...] time;`

`fastcgi_no_cache` Context: `http`, `server`, `location`	You may want to disable caching for requests that meet certain conditions. The directive accepts a series of variables. If at least one of these variables has a value (not an empty string and not `0`), this request will not be stored in cache. Syntax: `fastcgi_no_cache $variable1 [$variable2] [...];` Example: `fastcgi_no_cache $args_nocaching;`
`fastcgi_cache_bypass` Context: `http`, `server`, `location`	This directive functions in a similar manner to `fastcgi_no_cache`, except that it tells Nginx whether or not the request should be *loaded* from cache, if it can be (as opposed to deciding whether to *store* the request result in cache). Syntax: `fastcgi_cache_bypass $variable1 [$variable2] [...];` Example: `fastcgi_cache_bypass $cookie_bypass_cache;`

`fastcgi_cache_lock`, `fastcgi_cache_lock_timeout`, `fastcgi_cache_lock_age`
Context: `http`, `server`, `location`
	If set to `on`, `fastcgi_cache_lock` prevents repopulating the existing cache elements for the duration specified by `fastcgi_cache_lock_age` (`fastcgi_cache_lock_timeout` achieves the same result, except the response isn't cached).
	Example:
	`fastcgi_cache_lock on;`
	`fastcgi_cache_lock_timeout 10s;`

Directive	Description
`fastcgi_cache_revalidate` Context: `http`, `server`, `location`	When enabled, Nginx revalidates the expired cache items when instructed to do so by the `If-modified-since` and `If-none-match` headers. Syntax: `fastcgi_cache_revalidate on | off;` Default: off
`fastcgi_buffering`, `fastcgi_request_buffering` Context: `http`, `server`, `location`	Enables or disables the buffering of responses (or client requests, in the case of `fastcgi_request_buffers`) sent by a FastCGI backend. When disabled, Nginx forwards responses to the client synchronously. When enabled, responses are stored in buffers until the backend finishes sending the entire content and then sent to the client. Syntax: `fastcgi_buffering on | off;` Default: on
`fastcgi_buffers` Context: `http`, `server`, `location`	This directive sets the amount and size of buffers that will be used for reading the response data from the FastCGI application. Syntax: `fastcgi_buffers amount size;` Default: 8 buffers, 4 k or 8 k each, depending on the platform Example: `fastcgi_buffers 8 4k;`
`fastcgi_buffer_size` Context: `http`, `server`, `location`	This directive sets the size of the buffer for reading the beginning of the response from the FastCGI application, which usually contains simple header data. The default value corresponds to the size of 1 buffer as defined by the previous directive (`fastcgi_buffers`). Syntax: Size value Example: `fastcgi_buffer_size 4k;`